7. Las clases Java DatagramPacket y DatagramSocket

El objetivo de esta sesión práctica es mostrar el modo de funcionamiento de las clases Java para definir datagramas y s*ockets* de datagrama.

7.1 Introducción

Las redes actuales utilizan el *packet switching* para la transferencia de datos. Los datos se envuelven en paquetes que se transfieren desde un origen a un destino, donde se extraen de uno en uno los datos de uno o más paquetes para reconstruir el mensaje original.

Los nodos que se comunican a través de Internet utilizan principalmente dos protocolos:

- TCP - *Transsmision Control Protocol*
- UDP – *(Universal | User) Datagram Protocol*

El protocolo UDP – (User | Universal) Datagram Protocol - se utiliza para comunicaciones en la que no se garantiza una transmisión fiable (reliable). UDP no está orientado a conexión, por lo tanto no garantiza la entrega. UDP envía paquetes de datos independientes, denominados *datagramas*, desde una aplicación a otra.

El envío de datagramas es similar a enviar una carta a través del servicio postal: El orden de salida no es importante y no está garantizado, y cada mensaje es independiente de cualquier otro.

En las comunicaciones basadas en datagramas como las UDP, el paquete de datagrama contiene el número de puerto de su destino y UDP encamina el paquete a la aplicación apropiada

El API Java para UDP proporciona una abstración del “paso de mensajes", esto es, la forma más simple de comunicación entre ordenadores. Esto hace posible a un proceso emisor transmitir un único mensaje a un proceso receptor. Los paquetes independientes que contienen esos mensajes se denominan datagramas. En Java, el emisor especifica el destino usando un *socket* (una referencia indirecta a un puerto particular usada por el proceso receptor en la máquina receptora).

Un datagrama enviado mediante UDP es trasmitido desde un proceso emisor a un

proceso receptor sin reconocimiento o recomprobaciones. Si tiene lugar un fallo, el mensaje puede no llegar. Un datagrama es transmitido entre procesos cuando un proceso lo envía y otro proceso lo recibe. Cualquier proceso que necesite enviar o recibir mensajes debe en primer lugar crear un *socket* a un dirección de Internet y a un puerto local. Un servidor enlazará ese *socket* a un puerto servidor - uno que se hace conocido a los clientes de manera que puedan enviar mensajes al mismo. Un cliente enlaza su *socket* a cualquier puerto local libre. El método receptor devuelve la dirección de Internet y el puerto del emisor, además del mensaje, permitiendo a los receptores enviar una respuesta.

Las clases Java para establecer comunicaciones mediante datagramas son: DatagramPacket y DatagramSocket.

7.2 La clase DatagramPacket

La clase DatagramPacket proporciona un constructor que permite crear instancias de un array de bytes para: el mensaje, la longitud del mensaje, la dirección Internet y el puerto local del *socket* de destino, de la siguiente forma:

Los objetos del tipo DatagramPacket se pueden transmitir entre procesos cuando un proceso los envía y otro los recibe.

Esta clase proporciona otro constructor para usarlo cuando se recibe un mensaje. Sus argumentos especifican un array de bytes en el que recibir el mensaje y la longitud del array. Cuando se recibe un mensaje se pone en el DatagramPacket junto con su longitud, la dirección de Internet y el puerto del *socket* de envío.

Se puede obtener el mensaje del objeto DatagramPacket mediante el método getData(). Los métodos getPort() y getAddress() permiten obtener el puerto y la dirección Internet del objeto de tipoDatagramPacket.

El proceso receptor del mensaje tiene que especificar un array de bytes de un tamaño determinado en el cual recibir el mensaje, esto es, ha de predecir el *Tamaño del Mensaje*. Si el mensaje es muy grande para el array se trunca cuando llega. El protocolo IP subyacente permite longitudes de paquetes de más de 2^{16} bytes, que incluye tanto las cabeceras como los mensajes. Sin embargo, la mayoría de los entornos imponen una restricción en el tamaño a 8 kilobytes. Cualquier aplicación que necesite mensajes mayores que el máximo, debe fragmentarlos en pedazos de ese tamaño. Generalmente, una aplicación decidirá sobre un tamaño que no sea excesivamente grande pero que se adecue a su uso previsto.

7.3 La clase DatagramSocket

La clase DatagramSocket da soporte a s*ockets* para el envío y recepción de datagramas UDP.

Se proporciona un constructor que toma un puerto como argumento, para que sea usado por los procesos que necesitan usar un puerto particular. También se proporciona un constructor sin argumentos que permite al sistema escoger un puerto local libre. Estos constructores pueden lanzar una excepción del tipo SocketException si el puerto ya está en uso o si está reservado.

Esta clase cuenta con los siguientes métodos:

•send() y receive().

Estos métodos permiten transmitir datagramas entre un par de *sockets*. El argumento del send es una instancia de un DatagramPacket que contiene un mensaje y su destino. El argumento del receive es un objeto DatagramPacket vacío en el cual se pondrá el mensaje, su longitud y su origen. Tanto el método send() como el receive() pueden lanzar una IOException.

Las comunicaciones mediante datagramas de UDP usan envíos no bloqueantes (*non-blocking sends*) y recepciones bloqueantes (*blocking receives*). Las operaciones de envío retornan cuando estas han dado el mensaje a los protocolos IP o UDP subyacentes, los cuales son responsables de trasmitirlos a su destino. En la llegada, el mensaje es puesto en una cola por el *socket* que está asociado al puerto de destino. El mensaje puede ser recogido de la cola por una excepción o llamadas futuras de recepción (receive()) sobre ese *socket*. Los mensajes son descartados en el destino si ningún proceso tiene asociado un *socket* al puerto de destino. El método receptor (receive()) se bloquea hasta que se recibe un datagrama, a menos que se establezca un tiempo límite (time out) sobre el *socket*. Si el proceso que invoca al método receive() tiene otra tarea que hacer mientras espera por el mensaje, debería planificarse en un flujo de ejecución (*thread*) separado.

•setSoTimeout().

Este método permite establecer un tiempo de espera. Con un tiempo de espera establecido, el método receive() se bloqueará por el tiempo especificado y entonces lanzará una InterruptedIOException().

•connect().

Este método se utiliza para conectar a un puerto remoto particular y una dirección de Internet, en este caso el *socket* sólo es capaz de enviar y recibir mensajes desde esa dirección.